

Decision Tree

Decision trees are versatile Machine Learning algorithm that can perform both classification and regression tasks. They are very powerful algorithms, capable of fitting complex datasets.

We start by downloading the file from the web and loading it into R. This particular version of the file comes from Penn State.

Dataset can be downloaded from

https://onlinecourses.science.psu.edu/stat857/sites/onlinecourses.science.psu.edu.stat857/files/german_credit.csv

A popular and useful .csv file containing information on loan applicants at German banks is available from many sites on the web. The file contains 20 pieces of information on 1000 applicants.

The following code can be used to determine if an applicant is credit worthy and if he (or she) represents a good credit risk to the lender. Several methods are applied to the data to help make this determination.

1. Load the dataset `german_credit.csv` and view the dataset

```
#read data file
mydata= read.csv("german_credit.csv")
names(mydata)
```

```
## [1] "Creditability"      "Account.Balance"
## [3] "Duration.of.Credit..month." "Payment.Status.of.Previous.Credit"
## [5] "Purpose"            "Credit.Amount"
## [7] "Value.Savings.Stocks" "Length.of.current.employment"
## [9] "Instalment.per.cent" "Sex...Marital.Status"
## [11] "Guarantors"         "Duration.in.Current.address"
## [13] "Most.valuable.available.asset" "Age..years."
## [15] "Concurrent.Credits" "Type.of.apartment"
## [17] "No.of.Credits.at.this.Bank" "Occupation"
## [19] "No.of.dependents"    "Telephone"
## [21] "Foreign.Worker"
```

```
# Check attributes of data
str(mydata)
```

```
## 'data.frame':    1000 obs. of  21 variables:
## $ Creditability      : int  1 1 1 1 1 1 1 1 1 1 ...
## $ Account.Balance    : int  1 1 2 1 1 1 1 1 4 2 ...
## $ Duration.of.Credit..month. : int  18 9 12 12 12 10 8 6 18 24 ...
## $ Payment.Status.of.Previous.Credit: int  4 4 2 4 4 4 4 4 4 2 ...
## $ Purpose            : int  2 0 9 0 0 0 0 0 3 3 ...
## $ Credit.Amount      : int  1049 2799 841 2122 2171 2241 3398 1361 1098 3758 ...
## $ Value.Savings.Stocks : int  1 1 2 1 1 1 1 1 1 3 ...
## $ Length.of.current.employment : int  2 3 4 3 3 2 4 2 1 1 ...
## $ Instalment.per.cent : int  4 2 2 3 4 1 1 2 4 1 ...
## $ Sex...Marital.Status : int  2 3 2 3 3 3 3 3 2 2 ...
## $ Guarantors         : int  1 1 1 1 1 1 1 1 1 1 ...
## $ Duration.in.Current.address : int  4 2 4 2 4 3 4 4 4 4 ...
## $ Most.valuable.available.asset : int  2 1 1 1 2 1 1 1 3 4 ...
## $ Age..years.        : int  21 36 23 39 38 48 39 40 65 23 ...
## $ Concurrent.Credits : int  3 3 3 3 1 3 3 3 3 3 ...
## $ Type.of.apartment  : int  1 1 1 1 2 1 2 2 2 1 ...
## $ No.of.Credits.at.this.Bank : int  1 2 1 2 2 2 2 1 2 1 ...
## $ Occupation         : int  3 3 2 2 2 2 2 2 1 1 ...
## $ No.of.dependents   : int  1 2 1 2 1 2 1 2 1 1 ...
## $ Telephone          : int  1 1 1 1 1 1 1 1 1 1 ...
## $ Foreign.Worker     : int  1 1 1 2 2 2 2 2 1 1 ...
```

```
#Check number of rows and columns
dim(mydata)
```

```
## [1] 1000 21
```

Total number of records dataset 1000 and 21 attributes are used in the dataset

2. Make dependent variable as a factor (categorical)

```
#Make dependent variable as a factor (categorical)

mydata$Creditability = as.factor(mydata$Creditability)
```

3. Split data into training (70%) and validation (30%)

```
dt = sort(sample(nrow(mydata), nrow(mydata)*.7))
train<-mydata[dt,]
val<-mydata[-dt,]
nrow(train)
```

```
## [1] 700
```

This gives the train dataset have 700 records and 300 records are used for the testing

4. Construction of Decision Tree Model and visualize the decision tree

```
library(rpart)
library(rpart.plot)
library(RColorBrewer)
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
mtree <- rpart(Creditability~., data = train, method="class", control = rpart.control(minsplit = 20, minbuck
et = 7, maxdepth = 10, usesurrogate = 2, xval =10 ))
mtree
```

```

## n= 700
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 700 211 1 (0.30142857 0.69857143)
##    2) Account.Balance< 2.5 381 172 1 (0.45144357 0.54855643)
##      4) Duration.of.Credit..month.>=22.5 169 68 0 (0.59763314 0.40236686)
##        8) Value.Savings.Stocks< 3.5 147 53 0 (0.63945578 0.36054422)
##          16) Duration.of.Credit..month.>=47.5 28 3 0 (0.89285714 0.10714286) *
##          17) Duration.of.Credit..month.< 47.5 119 50 0 (0.57983193 0.42016807)
##            34) Purpose< 0.5 21 4 0 (0.80952381 0.19047619) *
##            35) Purpose>=0.5 98 46 0 (0.53061224 0.46938776)
##              70) Purpose>=1.5 82 34 0 (0.58536585 0.41463415)
##                140) Payment.Status.of.Previous.Credit< 1.5 15 2 0 (0.86666667 0.13333333) *
##                141) Payment.Status.of.Previous.Credit>=1.5 67 32 0 (0.52238806 0.47761194)
##                  282) Length.of.current.employment< 3.5 44 17 0 (0.61363636 0.38636364)
##                    564) Credit.Amount< 3128.5 16 2 0 (0.87500000 0.12500000) *
##                    565) Credit.Amount>=3128.5 28 13 1 (0.46428571 0.53571429)
##                      1130) Telephone>=1.5 12 3 0 (0.75000000 0.25000000) *
##                      1131) Telephone< 1.5 16 4 1 (0.25000000 0.75000000) *
##                        283) Length.of.current.employment>=3.5 23 8 1 (0.34782609 0.65217391)
##                          566) Purpose< 2.5 8 2 0 (0.75000000 0.25000000) *
##                          567) Purpose>=2.5 15 2 1 (0.13333333 0.86666667) *
##                            71) Purpose< 1.5 16 4 1 (0.25000000 0.75000000) *
##                              9) Value.Savings.Stocks>=3.5 22 7 1 (0.31818182 0.68181818) *
##                                5) Duration.of.Credit..month.< 22.5 212 71 1 (0.33490566 0.66509434)
##                                  10) Payment.Status.of.Previous.Credit< 1.5 18 4 0 (0.77777778 0.22222222) *
##                                  11) Payment.Status.of.Previous.Credit>=1.5 194 57 1 (0.29381443 0.70618557)
##                                    22) Purpose< 7 177 57 1 (0.32203390 0.67796610)
##                                      44) Credit.Amount< 1282 62 28 1 (0.45161290 0.54838710)
##                                        88) Credit.Amount>=668 49 22 0 (0.55102041 0.44897959)
##                                          176) Credit.Amount< 954.5 19 4 0 (0.78947368 0.21052632) *
##                                          177) Credit.Amount>=954.5 30 12 1 (0.40000000 0.60000000)
##                                            354) Credit.Amount>=1207.5 10 2 0 (0.80000000 0.20000000) *
##                                            355) Credit.Amount< 1207.5 20 4 1 (0.20000000 0.80000000) *
##                                              89) Credit.Amount< 668 13 1 1 (0.07692308 0.92307692) *
##                                              45) Credit.Amount>=1282 115 29 1 (0.25217391 0.74782609) *
##                                                23) Purpose>=7 17 0 1 (0.00000000 1.00000000) *
##                                                  3) Account.Balance>=2.5 319 39 1 (0.12225705 0.87774295) *

```

```

#Plot tree
plot(mtree)
text(mtree)

```

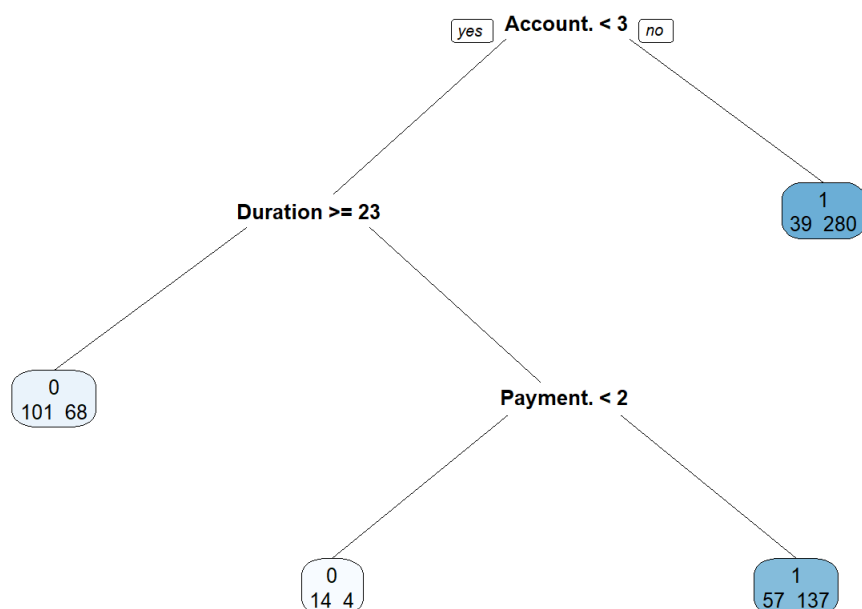


```
##
## Classification tree:
## rpart(formula = Creditability ~ ., data = train, method = "class",
##       control = rpart.control(minsplit = 20, minbucket = 7, maxdepth = 10,
##       usesurrogate = 2, xval = 10))
##
## Variables actually used in tree construction:
## [1] Account.Balance          Credit.Amount
## [3] Duration.of.Credit..month. Length.of.current.employment
## [5] Payment.Status.of.Previous.Credit Purpose
## [7] Telephone                Value.Savings.Stocks
##
## Root node error: 211/700 = 0.30143
##
## n= 700
##
##      CP nsplit rel error  xerror   xstd
## 1 0.078199     0  1.00000 1.00000 0.057539
## 2 0.047393     2  0.84360 0.89100 0.055575
## 3 0.037915     3  0.79621 0.85782 0.054902
## 4 0.016114     4  0.75829 0.85782 0.054902
## 5 0.012638     9  0.67773 0.88626 0.055481
## 6 0.010000    17  0.54976 0.85782 0.054902
```

The easiest way to plot a tree is to use `rpart.plot`. This function is a simplified front-end to the workhorse function `prp`, with only the most useful arguments of that function. Its arguments are defaulted to display a tree with colors and details appropriate for the model's response with the total of nodes in each classification

6. Pruning the Decision Tree and visualize the pruned tree

```
bestcp <- mtree$cpstable[which.min(mtree$cpstable[, "xerror"]), "CP"]
# Prune the tree using the best cp.
pruned <- prune(mtree, cp = bestcp)
#Plot pruned tree
prp(pruned, box.palette = "Blues", faclen = 0, cex = 0.8, extra = 1)
```



Validation of decision tree using the 'Complexity Parameter' and cross validated error. To validate the model we use the `printcp` and `plotcp` functions. 'CP' stands for Complexity Parameter of the tree. Prune the tree to avoid any overfitting of the data. Pruned Tree is visualized with `prp` function

7. print the confusion matrix (training data)

```

conf.matrix <- table(train$Creditability, predict(pruned,type="class"))
rownames(conf.matrix) <- paste("Actual", rownames(conf.matrix), sep = ":")
colnames(conf.matrix) <- paste("Pred", colnames(conf.matrix), sep = ":")
print(conf.matrix)

```

```

##
##          Pred:0 Pred:1
## Actual:0      115     96
## Actual:1       72    417

```

Printing the Confusion matrix for training dataset

8. Ploting ROC Curve

```

#Scoring
library(ROCR)

```

```

## Loading required package: gplots

```

```

##
## Attaching package: 'gplots'

```

```

## The following object is masked from 'package:stats':
##
##      lowess

```

```

val1 = predict(pruned, val, type = "prob")
#Storing Model Performance Scores
pred_val <- prediction(val1[,2],val$Creditability)
# Calculating Area under Curve
perf_val <- performance(pred_val,"auc")
perf_val

```

```

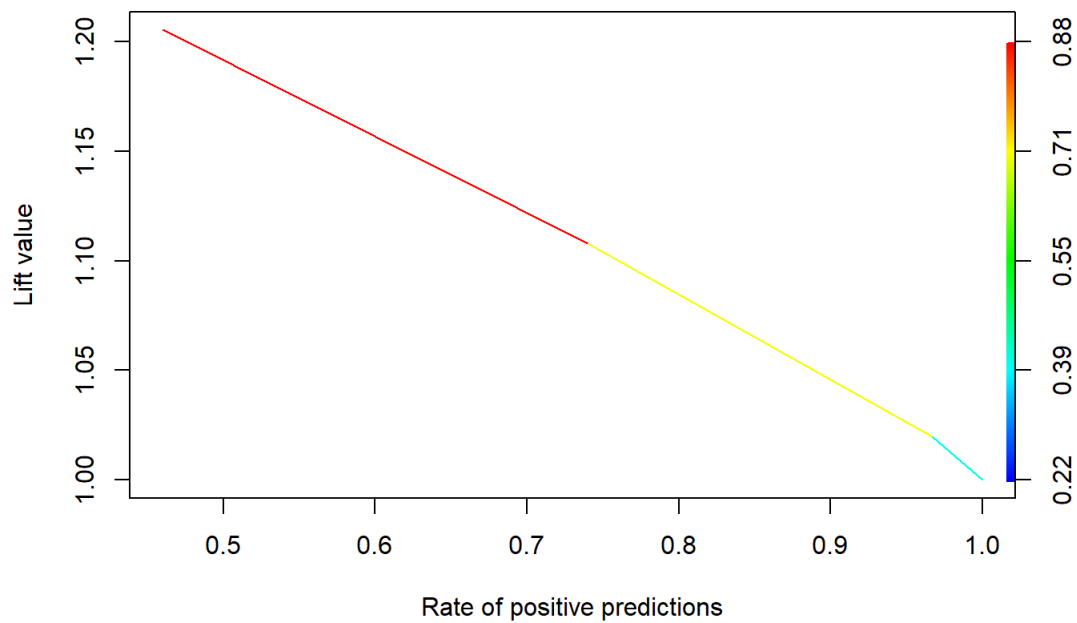
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.6944725
##
##
## Slot "alpha.values":
## list()

```

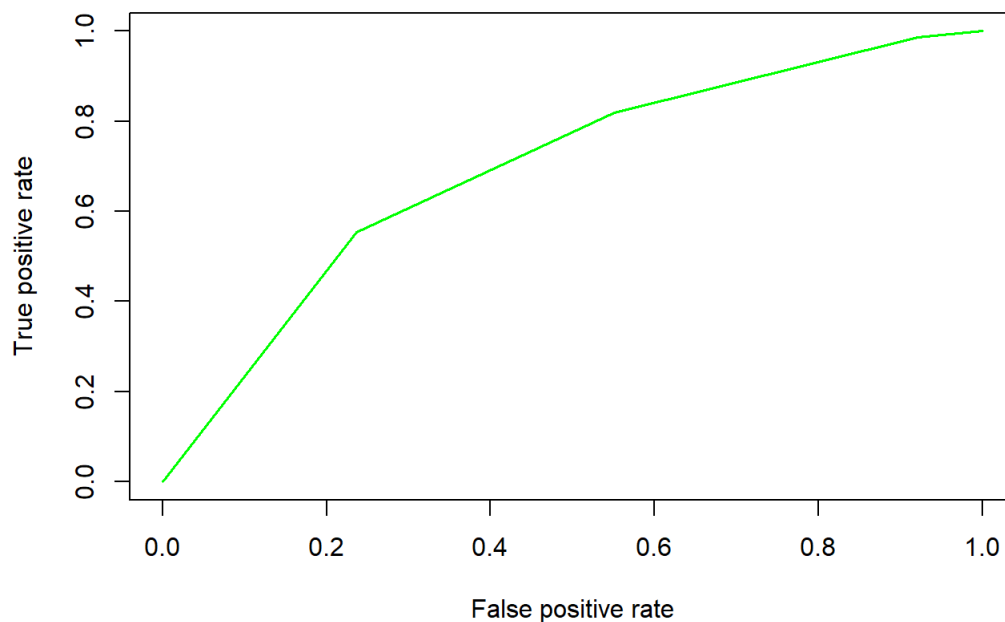
```

# Plotting Lift curve
plot(performance(pred_val, measure="lift", x.measure="rpp"), colorize=TRUE)

```



```
# Calculating True Positive and False Positive Rate
perf_val <- performance(pred_val, "tpr", "fpr")
#Plot the ROC curve
plot(perf_val, col = "green", lwd = 1.5)
```



```
#Calculating KS statistics
ks1.tree <- max(attr(perf_val, "y.values")[[1]] - (attr(perf_val, "x.values")[[1]]))
ks1.tree
```

```
## [1] 0.3185473
```

ROC Curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds. It also represents the rate of positive prediction with respect to lift value

9. Visualize the Tree using Random Forest Algorithm


```
library(randomForest)
```

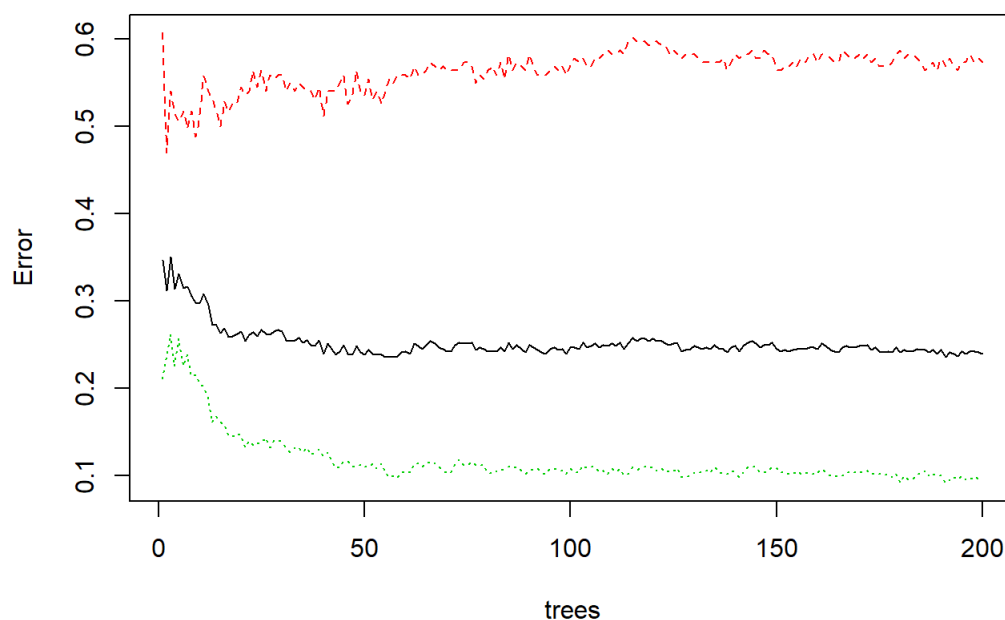
```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##  
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:rattle':  
##  
## importance
```

```
rf50 <- randomForest(Creditability ~., data = train, ntree=200, importance=T, proximity=T)  
plot(rf50, main="")
```



```
rf50
```

```
##  
## Call:  
## randomForest(formula = Creditability ~ ., data = train, ntree = 200, importance = T, proximity = T)  
  
## Type of random forest: classification  
## Number of trees: 200  
## No. of variables tried at each split: 4  
##  
## OOB estimate of error rate: 24%  
## Confusion matrix:  
## 0 1 class.error  
## 0 90 121 0.57345972  
## 1 47 442 0.09611452
```

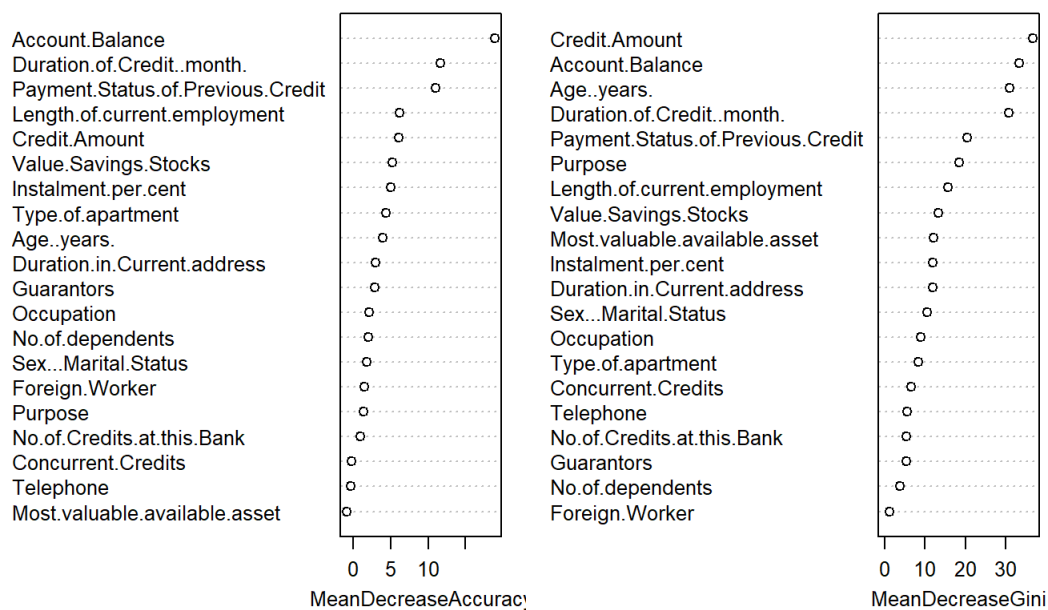
```
Test50_rf_pred <- predict(rf50, val, type="class")  
table(Test50_rf_pred, val$Creditability)
```

```
##
## Test50_rf_pred    0    1
##                0  35  17
##                1  54 194
```

```
importance(rf50)
```

```
##                0                1 MeanDecreaseAccuracy
## Account.Balance      17.0345567 12.1308594          18.9900629
## Duration.of.Credit..month.  5.5089601  9.4960792          11.6759057
## Payment.Status.of.Previous.Credit 8.4896046  7.6564677          11.0479730
## Purpose              0.9278426  1.0862223           1.3528053
## Credit.Amount        1.5948165  5.2779408           6.0318557
## Value.Savings.Stocks   4.7468597  2.6308096           5.2586942
## Length.of.current.employment  3.1965490  5.1314031           6.1695638
## Instalment.per.cent    1.7833825  4.8163537           5.0454649
## Sex...Marital.Status   1.3760454  1.1639996           1.8079605
## Guarantors            0.4995824  2.9017137           2.8675793
## Duration.in.Current.address -1.2837234  4.7582872           2.9648797
## Most.valuable.available.asset -2.7493004  1.1553206          -0.9086577
## Age..years.           2.2931555  3.1741150           3.9239554
## Concurrent.Credits     -2.4882521  1.8066010          -0.2782075
## Type.of.apartment      0.3838093  4.7472345           4.3395388
## No.of.Credits.at.this.Bank -2.8823896  2.8845424           0.9001578
## Occupation             0.9998946  1.8332942           2.1286349
## No.of.dependents       0.1112932  2.2591425           1.9868938
## Telephone             0.4062730 -0.9809385          -0.4053036
## Foreign.Worker         0.8270148  1.2119717           1.4913185
##                MeanDecreaseGini
## Account.Balance      33.263108
## Duration.of.Credit..month.  30.742653
## Payment.Status.of.Previous.Credit 20.392796
## Purpose              18.477091
## Credit.Amount        36.749883
## Value.Savings.Stocks  13.265203
## Length.of.current.employment  15.735681
## Instalment.per.cent    11.895126
## Sex...Marital.Status   10.473137
## Guarantors            5.379316
## Duration.in.Current.address 11.885468
## Most.valuable.available.asset 12.181252
## Age..years.           30.905994
## Concurrent.Credits     6.633364
## Type.of.apartment      8.307406
## No.of.Credits.at.this.Bank 5.471677
## Occupation             8.856564
## No.of.dependents       3.784461
## Telephone             5.557268
## Foreign.Worker         1.304793
```

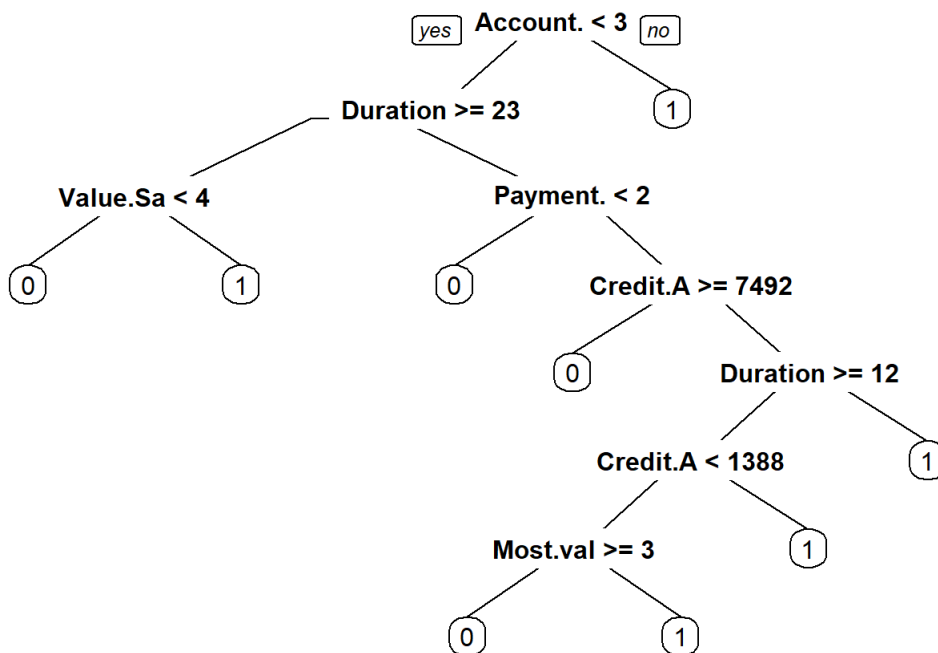
```
varImpPlot(rf50, main="", cex=0.8)
```



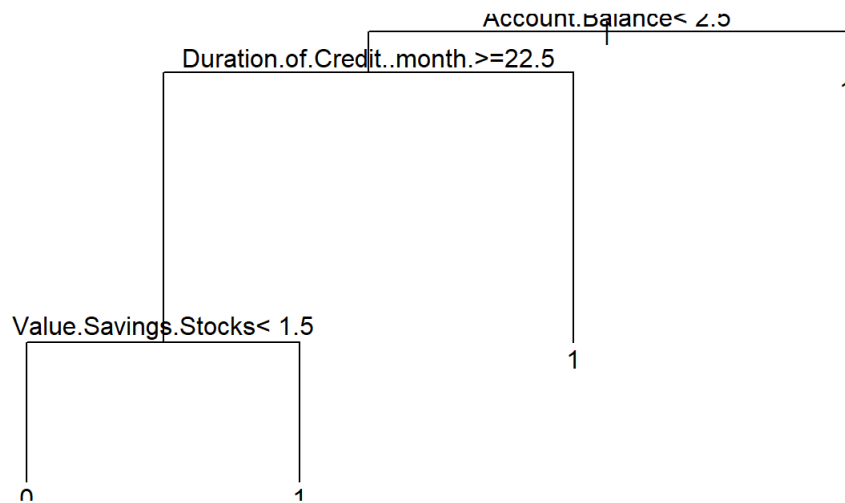
Random Forest is one such very powerful ensembling machine learning algorithm which works by creating multiple decision trees and then combining the output generated by each of the decision trees. Number of trees is 200 and number of variables tried at each split is 4 in this case. Error rate is 23.43%. Importance parameters are visualized with respect to accuracy and gini measure

10. Visualize the Tree using CART Model

```
# CART model
latlontree = rpart(mydata$Creditability~., data= mydata)
# Plot the tree using prp command defined in rpart.plot package
prp(latlontree)
```



```
latlontree = rpart(mydata$Creditability~., data= mydata,minbucket=50)
plot(latlontree)
text(latlontree)
```



A Decision Tree is a supervised learning predictive model that uses a set of binary rules to calculate a target value. It is used for either classification (categorical target variable) or regression (continuous target variable). Hence, it is also known as CART (Classification & Regression Trees). CART model is visualized with respect to creditability and all other parameters using rpart function

Conclusion

Various Tree visualization like Regression Tree, Random Forest and CART Models were analysed